Planification de mission pour une patrouille de micro-drones

Pierre-Selim Huard, Nicolas Barnier, Pascal Brisset, Gérard Verfaillie

Planification de mission pour une patrouille de micro-drones

Pierre-Selim Huard[1], Nicolas Barnier[1], Pascal Brisset[1], Gérard Verfaillie[2]

[1] École Nationale de l'Aviation Civile
7, avenue Édouard Belin, 31000 Toulouse
{huard, barnier, brisset}@recherche.enac.fr

[2] Office Nationale d'Études et de Recherches Aérospatiales
Département Commande des Systèmes et Dynamique du Vol,
2, avenue Édouard Belin, 31000 Toulouse
gerard.verfaillie@onera.fr

Résumé : Le problème de planification de mission en ligne pour une patrouille de mini-drones est un problème d'optimisation fortement combinatoire. Il s'agit d'affecter l'ensemble des objectifs de la mission aux drones de la patrouille et d'ordonner leur exécution pour que la patrouille finisse l'intégralité de la mission au plus tôt. Pour faire face aux incertitudes liées à l'exécution de la mission, les solutions sont recalculées de façon réactive si nécessaire pendant la mission.

Dans cet article nous présentons un système complet de planification de mission en ligne sous contraintes pour une patrouille de mini-drones. Nous proposons un modèle du problème en ligne sous forme d'un CSP qui est résolu à l'aide de la librairie de contraintes FaCiLe.

Mots-clés : Planification de mission, Ordonnancement, Système multi-agents, Système hybride réactif-délibératif.

1 Introduction

Le problème de planification de mission appliqué à une patrouille de drones consiste à répartir les objectifs à réaliser entre les différents drones de la patrouille et à construire un plan (suite d'actions) pour chaque drone qui permette de réaliser l'ensemble des objectifs qui lui ont été alloués. De plus on cherche à minimiser le temps total d'exécution de la mission, c'est-à-dire le temps mis par le drone finissant en dernier pour effectuer la mission. Cela nous permet de garder le plus de temps disponible[1] pour réagir aux incertitudes. Le système doit ainsi fournir à chaque véhicule son plan d'actions, mais aussi réagir aux événements tels que l'échec de la réalisation d'une tâche ou une modification du vent pouvant modifier les temps de parcours, ce qui nécessite de recalculer de nouveaux plans (en ligne) pendant l'exécution de la mission.

2 Contexte

2.1 Description du problème

On souhaite réaliser les missions des compétitions de drones MAV, EMAV et IMAV[2] avec trois types d'objectifs différents : largage, identification et localisation (voir figure 1). Afin de réaliser un largage, un drone de la patrouille doit lâcher une bille de peinture sur une cible dont les coordonnées sont connues. L'identification d'une cible revient à identifier le symbole peint sur une cible au sol dont les coordonnées sont connues. Enfin la localisation consiste à trouver les coordonnées d'une cible dans une zone rectangulaire connue. Ces missions « jouets » représentent aussi bien une mission de type militaire où l'on envoie une patrouille pour effectuer de la reconnaissance et des largages en zone ennemie, qu'une mission de type

[1] Les batteries utilisées actuellement sur les micro-drones donnent une autonomie d'environ 30 minutes.

[2] EMAV = European Micro Air Vehicle Conference and Flight Competition, (I)MAV = US-European Competition and Workshop on Micro Air Vehicle

civil tel que la localisation de survivants, le largage de vivres (à ces derniers), ou la surveillance de feux de forêt.

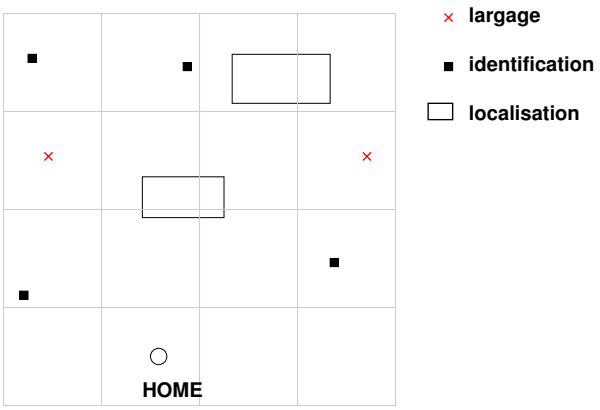

FIG. 1 – Exemple d'une mission avec 4 cibles à identifier, 2 cibles de largage et 2 à localiser.

Tous les drones de la patrouille décollent de la même base (le point $HOME$). Ils évoluent ensuite dans la zone de vol afin de réaliser les identifications, largages et localisations spécifiés pour la mission. Chaque drone effectue ses actions de façon indépendante des autres ce qui signifie que les drones ne coopèrent pas entre eux pour effectuer une tâche précise à plusieurs. Une fois qu'il ne reste plus d'objectif à affecter, les drones ayant fini la réalisation de leurs buts (objectifs leur ayant été attribués) atterrissent au point $HOME$.

2.2 Travaux connexes

Le problème de planification de mission que l'on présente comporte des similitudes avec les problèmes classiques de *tournées de véhicules* (VRP) et d'*ordonnancement d'atelier* (JobShop) avec temps de transition : on doit allouer tous les objectifs de la mission à réaliser, et ordonnancer l'exécution de ces derniers afin de créer les plans d'actions de chacun des drones. Une des principales différences entre VRP et JobShop (Beck *et al.* (2003)) est que pour le premier on cherche généralement à minimiser la somme du coût de toutes les tournées et que pour le second on cherche à minimiser la date de fin.

Caseau & Laburthe (1997) présentent une méthode complète de résolution du problème du voyageur de commerce pour des petites instances du problème. Chaque solution du problème est une liste chaînée représentant la suite des villes à visiter. Rousseau, Gendreau et Pesant utilisent un modèle similaire pour définir les solutions d'un problème de tournées de véhicules dans (Rousseau *et al.* (2002)). Le cas des instances de grande taille est résolu à l'aide de métaheuristiques ou de recherche locale telles que les méthodes de recherche locale à grand voisinage (Shaw (1998)).

Pour le problème de planification de mission pour une patrouille de mini-drones, le temps de réalisation de chaque tâche dépend de la vitesse sol du drone la réalisant et donc de la vitesse du vent. On parle de problème asymétrique.

3 Modélisation

3.1 Modèle CSP

3.1.1 Les données

Soit N le nombre de cibles et M le nombre de drones de la patrouille. On définit une tâche pour chacune des différentes cibles et une tâche de début et de fin supplémentaires pour chaque drone. La tâche de début correspond soit au décollage soit à la fin de la tâche courante pour le cas d'une planification en ligne (en cours de mission) et la tâche de fin correspond à l'atterrissage. Par convention les N premières tâches correspondent aux cibles, et pour chaque drone $k \in [1, M]$ la tâche $N + k$ correspond à la première tâche de son plan et la tâche $N + M + k$ à son atterrissage.

FIG. 2 – Numérotation des tâches.

On note $TIME_{k,i,j}$ le temps mis par le drone k pour réaliser la tâche j après la tâche i. Ce temps inclus le temps de déplacement (transition) pour aller de i à j. Pour chaque tâche, on définit son type $T_i \in \{Id, Dr, Se, To, La\}$ avec Id pour identification, Dr pour largage, Se pour localisation, To pour décollage et La pour atterrissage.

Concernant la charge utile, on note $BALL_k \in \mathbb{N}$ le nombre de billes que peut larguer le drone k et $VIDEO_k \in \{true, false\}$ la présence d'une caméra dans le drone k.

3.1.2 Les variables

On note $route_i \in [1, M]$ la variable qui représente le drone auquel est affectée la tâche $i \in [1, N + 2M]$. Les tâche d'indice $N + 1$ à $N + 2M$ sont déjà affectées, il s'agit des tâches de décollage (voir équation (1)) et d'atterrissage (voir équation (2)).

$$\forall k \in [1, M] \qquad route_{N+k} = k \tag{1}$$

$$\forall k \in [1, M] \qquad route_{N+M+k} = k \tag{2}$$

Afin de représenter une solution, on doit trouver dans quel ordre chaque drone effectue l'ensemble des tâches qui lui est affecté. Pour cela, on modélise chaque plan comme une liste de tâches. On appelle $next_i \in [1, N + 2M]$ la tâche effectuée après la tâche i. De façon symétrique on définit $prev_i \in [1, N + 2M]$ la tâche précédant la tâche i. Par convention, on fixe $prev_{N+k} = N + M + k$ et $next_{N+M+k} = N + k$ pour tout $k \in [1, M]$.

Sur la figure 3 on peut voir une solution pour la mission de la figure 1. Sur cette figure les flèches représentent les relations de type successeur (les variables $next$) : par exemple $prev_2 = 5$ et $next_5 = 2$.

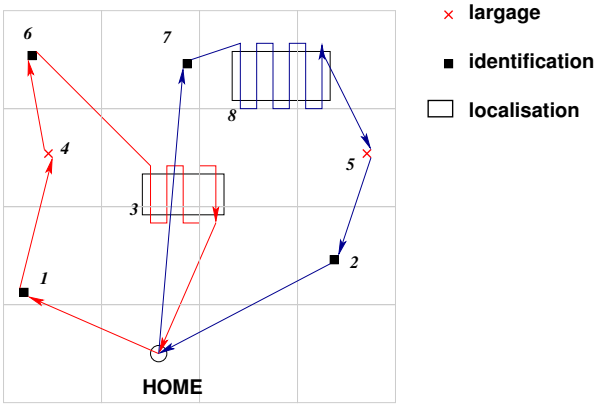

FIG. 3 – Exemple de solution pour une mission à 8 cibles et 2 drones.

Les variables $prev$, $next$ et $route$ sont liées entre autres par les contraintes suivantes :

$$next_i = j \quad \Leftrightarrow \quad prev_j = i \tag{3}$$

$$next_i = j \quad \Rightarrow \quad route_i = route_j \tag{4}$$

$$next_i \neq i \tag{5}$$

$$Alldiff(next) \tag{6}$$

$$Alldiff(prev) \tag{7}$$

Les contraintes (6) et (7) représentent le fait que chaque tâche est effectuée une seule fois.

3.1.3 Élimination des cycles : NoSubTour

Les variables de décisions sont les $(prev_i)_i$ (ou $(next_i)_i$). Les solutions valides sont les solutions ne contenant pas de cycle interne. Pour cela on utilise la contrainte `NoSubTour`, que l'on peut trouver par exemple dans les travaux de Caseau & Laburthe (1997); Rousseau *et al.* (2002). La contrainte `NoSubTour` peut se formuler de la façon suivante : à partir de n'importe quelle tâche effectuée par un drone on atteint la tâche d'atterrissage en composant *« l'opérateur next »* un nombre fini de fois (e.g. inférieur à la taille du plan de ce drone).

$\forall i \in [1, N{+}2M], \exists k \leq \sum_j (route_j = route_i)$ tel que $i_k = N{+}M{+}route_i$ avec $\left\{ \begin{array}{rcl} i_1 & = & next_i \\ i_{n+1} & = & next_{i_n} \end{array} \right.$

Afin de propager cette contrainte, pour chaque « chaîne partielle de tâches » définie par l'ensemble des variables $(next_i)_i$, on sauvegarde explicitement la première $(start)$ et la dernière (end) tâche. Lorsque deux chaînes sont fusionnées, on met à jour les informations concernant la première et la dernière tâche de cette chaîne. Ensuite, on enlève la tâche end du domaine de la variable $prev$ correspondant à la tâche $start$, i.e. $prev_{start} \neq end$, pour interdire tout cycle. De façon symétrique, à l'instanciation de $next_i = j$ on a $next_{end} \neq start$ (voir figure 4).

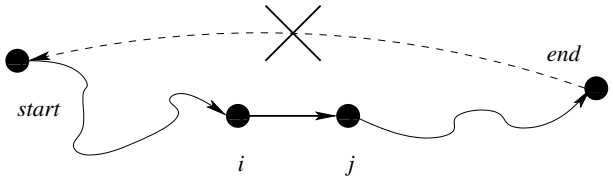

FIG. 4 – La contraintes `NoSubTour` avec $next_i = j$

3.1.4 Les contraintes de capacité

Seuls les drones ayant une caméra peuvent effectuer les tâches d'identification et de localisation.

$$\forall i \in [1, M], \forall k \in [1, N] \text{ tel que } [(T_k = Se) \vee (T_k = Id)] \text{ et } \neg VIDEO_i \text{ alors } route_k \neq i \quad (8)$$

Un drone ne peut pas effectuer plus de largages que le nombre total de billes de peinture qu'il possède, c'est-à-dire $\forall i \in [1, M]$:

$$\sum_{i=1}^{N+2M} ((T_i = Dr) \cdot (route_i = k)) \leq BALL_k \quad (9)$$

3.1.5 Le critère d'optimisation

Pour chaque drone i le temps d'exécution de la mission (décollage, atterrissage et exécution des objectifs) est la somme du temps de chaque tâche effectuée par le drone i :

$$cost_k = \sum_{i=1}^{N+2M} TIME_{k,i,next_i} \cdot (route_i = k)$$

Ce temps d'exécution doit être inférieur au temps limite d'exécution de la mission. Ce temps limite est imposé par l'autonomie des batteries embarquées ($ENDURANCE_k$) sur les drones. On pose donc la contrainte suivante :

$$\forall i \in [1, M], cost_i \leq ENDURANCE_k \quad (10)$$

On cherche à minimiser le temps total d'exécution de la mission, c'est-à-dire le maximum des temps d'exécution de chaque drone :

$$\min \max_{k \in [1,M]} cost_k \quad (11)$$

4 Résolution

Les deux méthodes que nous présentons dans cette section ont été mises en œuvre à l'aide de la librairie de programmation par contraintes FaCiLe (Barnier (2002)).

4.1 Résolution complète

Dans cette section nous décrivons en détail les mécanismes de séparation et d'évaluation (Branch & Bound) utilisés pour la résolution complète du problème de planification de mission en ligne pour une patrouille de drones. FaCiLe fournit un but d'optimisation qui pose dynamiquement, à chaque fois qu'une nouvelle solution de coût $bestcost$ est trouvée, la contrainte sur le coût : $cost_k < bestcost$.

4.1.1 Séparation

La stratégie classique de séparation (Baptiste *et al.* (2001)) pour les problèmes tels que JobShop ou VRP est de réaliser d'abord l'allocation de ressources, c'est-à-dire affecter les tâches aux drones (variables $route_i$) puis de calculer l'ordonnancement des tâches de chaque drone (variables $prev$ et $next$) .

Afin de ne pas parcourir des solutions de trop mauvais coût en premier e.g. toutes les tâches affectées à un seul drone, on instancie les valeurs du domaine de $route_i$ de façon à minimiser le maximum de la durée de vol des drones. Intuitivement on répartit le travail à faire entre tous les drones disponibles.

Après cette phase d'allocation, on sélectionne en premier les variables $(next_i)$ correspondant au drone ayant la durée de vol la plus grande afin d'échouer le plus tôt possible dans l'arbre de recherche. Parmi ces variables, on sélectionne en premier la variable dont le domaine a la plus petite taille. Cet ordre de sélection permet de gagner en temps de recherche car on arrive à évaluer plus rapidement la borne inférieure du coût de chaque drone.

4.1.2 Évaluation

On peut minorer la durée de vol pour chaque drone k par :

$$\mathcal{P}_k = \sum_{i \in [1, N+2M]} (route_i = k) \min_{j \in dom(prev_i)} TIME_{k,j,i} \tag{12}$$

$$\mathcal{N}_k = \sum_{i \in [1, N+2M]} (route_i = k) \min_{j \in dom(prev_i)} TIME_{k,i,j} \tag{13}$$

Ces minorants $\mathcal{P}_k$ et $\mathcal{N}_k$ peuvent être utiliser afin de réduire le domaine des variables $(prev_i)$ et $(next_i)$. En effet, si le temps pour effectuer la tâche i après la tâche j est trop important, le drone n'aura pas le temps de finir la mission avant $bestcost$. Ce qui se traduit par les contraintes suivantes :

$\forall i \in [1, N + 2M], \forall j \in dom(prev_i)$

$$\left(TIME_{k,j,i} + \mathcal{P}_k - \min_{t \in dom(prev_i)} TIME_{k,t,i} > bestcost \right) \Rightarrow (prev_i \neq j) \tag{14}$$

$\forall i \in [1, N + 2M], \forall j \in dom(next_i)$

$$\left(TIME_{k,i,j} + \mathcal{N}_k - \min_{t \in dom(next_i)} TIME_{k,i,t} > bestcost \right) \Rightarrow (next_i \neq j) \tag{15}$$

4.2 Recherche locale à grand voisinage

Dans cette section nous décrivons une méthode de recherche locale à grand voisinage (Shaw (1998)) que nous utilisons pour résoudre le problème de planification en ligne. Cette méthode hybride *recherche locale* et *résolution complète*. Le principe de la recherche locale à grand voisinage est de construire une solution partielle en détruisant une partie de la solution courante, et à partir de cette solution partielle de générer un grand voisinage de la solution courante en explorant l'ensemble des solutions possibles que l'on peut construire à partir de la solution partielle.

$S \leftarrow$ Solution initiale
Tant que Condition d'arrêt est fausse **Faire**
  $S^* \leftarrow$ Solution partielle à partir de S
  $V(S^*) \leftarrow$ Voisinage construit à partir de S^*
  $S \leftarrow$ Meilleur élément de $V(S^*)$
Fin tant que

Dans le cas de la programmation par contraintes et plus précisement pour la planification de mission en ligne la destruction se fait simplement en gardant une instanciation partielle de la solution courante. Pour cela il suffit de sélectionner un nombre x de tâches dont on défait l'instanciation (voir figure 5) des variables correspondantes ($route$, $prev$, et $next$). L'exploration du voisinage est alors faite en utilisant la méthode complète de séparation et évaluation présentée précédemment.

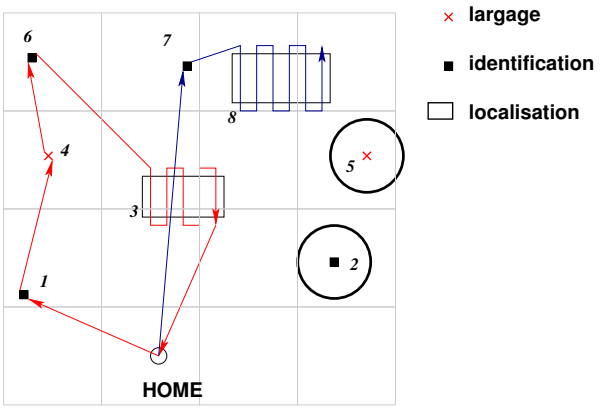

FIG. 5 – Exemple d'instantiation partielle pour la solution de la figure 3 où l'on a selectionné les cibles 2 et 5.

5 Intégration système

Pour prendre en compte les incertitudes liées au succès de la réalisation des actions lors de l'exécution des missions d'observations, il peut être nécessaire de calculer *en ligne* les plans en fonction du déroulement de l'exécution de la mission. La planification en ligne implique de fortes contraintes sur les temps de calcul des algorithmes de résolution : en effet, à tout moment de l'exécution de la mission, une action (ou un plan entier) doit être disponible pour chaque drone - un drone à voilure fixe ne peut pas s'arrêter pour laisser le temps au système de calculer une nouvelle décision.

Pour pallier ce problème on trouve différentes approches. Certaines donnent la priorité à la satisfaction des contraintes temporelles telles que les règles de décision, les algorithmes gloutons (Ostergaard *et al.* (2001)) ou encore la recherche locale. D'autres approches donnent la priorité à la qualité des solutions trouvées quitte à mettre en attente le système en attendant les résultats de la planification (Muscettola *et al.* (1998)). Ces deux approches ont pour défaut soit de ne pas chercher une solution de meilleure qualité même si le temps l'avait permis, soit de prendre le risque de perdre des opportunités dues au temps d'attente imposé au système.

La solution algorithmique à ce paradoxe est l'approche *anytime* (Zilberstein (1996)), terme introduit par Dean & Bobby (1988), où l'on garantit de trouver une solution dans une certaine limite de temps et où l'on améliore ensuite progressivement cette solution. Cette approche est un compromis entre les deux premières approches. Elle permet de privilégier un temps de réponse court pour la première réponse, et les améliorations successives permettent de trouver de meilleures solutions que la première tout en respectant la contrainte de temps fixée par le système.

L'exécution réelle de la mission est susceptible de remettre souvent en cause les hypothèses des algorithmes de planification telles que le temps de parcours des drones et la réussite des actions. Le système doit pouvoir s'adapter aux mises-à-jour de l'environnement et aux échecs. C'est pourquoi nous proposons d'utiliser une architecture hybride réactive et délibérative.

5.1 Modèle réactif

Lemaître & Verfaillie (2007) présentent un modèle d'architecture hybride réactive et délibérative à trois agents. Nous présentons ici une architecture fonctionnant suivant ce modèle avec trois agents :

- L'*environnement* envoie des événements à la tâche réactive en fonction de l'état de chaque drone.
- La *tâche réactive* reçoit les messages de l'environnement. En fonction de ces messages, la partie réactive peut relancer ou arrêter la tâche délibérative. Elle peut ensuite utiliser les délibérations fournies par la tâche délibérative pour envoyer des ordres aux drones, ou, si cette délibération n'est plus valable, c'est elle qui fournira une décision au drone concerné (à l'aide d'une régle de décision).
- La *tâche délibérative* fournit de façon anytime des plans d'actions pour chaque drone à la tâche réactive. Cette tâche est interruptible par la tâche réactive.

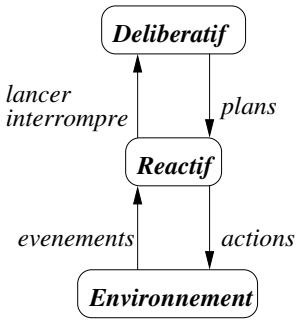

FIG. 6 – Architecture réactive.

Cette architecture offre des garanties en termes de décision : on est sûr que la tâche réactive fournira toujours une décision, soit celle calculée par la tâche délibérative si celle-ci est fournie à temps et est toujours cohérente avec l'état actuel de la patrouille de drone, soit celle fournie par la tâche réactive elle-même.

5.1.1 L'environnement

L'environnement envoie les messages suivants pour chaque drone :

- `READY`, le drone est prêt à décoller.
- `FLYING`, le drone a réussi son décollage et peut commencer la mission.
- `SUCCESS`, le drone a réussi la tâche qu'il était en train de faire.
- `LOST_AC`, le drone a été perdu (crash ou perte de communication).
- `FAIL`, le drone n'a pas réussi la tâche courante.
- `LANDED`, le drone vient d'atterrir.

L'environnement peut recevoir les ordres suivants pour chaque drone :

- `TOFF`, l'environnement sélectionne la prodécure décollage du plan de vol du drone.
- `EIGHT(WP`, l'environnement envoie les coordonnées de `WP`, et sélectionne la procédure identification[3]. de la cible `WP` du plan de vol du drone.
- `DROP(WP1)`, l'environnement envoie les coordonnées de `WP1` et sélectionne la prodécure de largage du plan de vol du drone.
- `SCAN(WP1, WP2)` l'environnement envoie les coordonnées de `WP1` et `WP2`, puis sélectionne la prodécure de balayage du plan de vol du drone (la zone à balayer est définie par les coins sud-ouest et nord-est).
- `LAND` sélectionne la prodécure atterrissage du plan de vol du drone.

5.1.2 La tâche réactive

La tâche réactive maintient à jour une liste des objectifs restant en attente de traitement que l'on notera *pendings*. À chaque fois qu'un drone commence une action, on met à jour la liste. La tâche réactive est conçue de façon à pouvoir effectuer la mission dans le cas où la tâche délibérative n'arrive pas à fournir de

[3] La procédure d'identification de `WP` est effectué en parcourant un 8 centré sur `WP`

délibération valide au moment de l'exécution. Pour définir la tâche réactive, il faut définir son comportement pour tous les événements qu'elle reçoit.

Au lancement du système, la tâche réactive lit les données de la mission, lance une tâche délibérative avec l'état initial du système, et attend de recevoir des événements de l'environnement. Elle réagit à ses événements de la manière suivante :

READY

À la réception de l'événement READY, la tâche réactive renvoie le message TOFF (décollage) à l'environnement.

FLYING

À la réception du message FLYING, la tâche réactive fournit si possible la délibération de la tâche délibérative, sinon elle choisit un objectif à effectuer en fonction des ressources disponibles et l'envoie à l'environnement. Dans les deux cas, l'objectif sélectionné est supprimé de la liste *pendings*. La tâche réactive interrompt la tâche délibérative, et relance une nouvelle tâche de décision.

FAIL

À la réception de l'événement FAIL, la tâche réactive replace l'objectif en cours de réalisation par le drone dans la liste *pendings* et interrompt la tâche délibérative en cours. Aucune décision n'étant disponible immédiatement, la tâche réactive sélectionne elle-même l'objectif à réaliser parmi les objectifs réalisables pour le drone.

SUCCESS

À la réception du message SUCCESS, la tâche réactive fournit si possible la délibération de la tâche délibérative, sinon elle choisit un objectif à effectuer en fonction des ressources disponibles et l'envoie à l'environnement. Dans les deux cas, l'objectif sélectionné est supprimé de la liste *pendings*. La tâche réactive interrompt la tâche délibérative en cours, et en relance une nouvelle (avec la deadline du plan en cours d'exécution).

LOST_AC

À la récéption de l'événement LOST_AC, la tâche réactive retire le drone de la liste des drones de la patrouille et replace l'objectif qu'il était en train de réaliser dans la liste *pendings*.

LANDED

À la réception de l'événement LANDED, la tâche réactive retire le drone de la liste des drones de la patrouille.

5.2 Intégration au système Paparazzi

Paparazzi (Brisset *et al.* (2006)) est un système *Open-Source*[4] de mini-drones. La station sol du système Paparazzi fonctionne de façon distribuée (Figure 7).

Les différents agents de la station sol communiquent entre eux à l'aide d'un bus logiciel *Ivy* (Buisson *et al.* (2002)). Les trois agents principaux *link*, *server* et *gcs* fonctionnent ainsi :

- L'agent *link* se charge des communications avec les périphériques. Il traduit les messages entre le bus et le matériel (modems), et permet de communiquer avec plusieurs drones.
- L'agent *server* écoute les messages venant de link et redistribue les informations de façon synthétique aux autres agents tels que les interfaces graphiques. Cet agent gère les fichiers de configuration (plans de vol, configuration des avions, etc.) et permet de les rendre disponibles sur le bus *Ivy*. Il calcule aussi des informations relatives à l'*environnement* (estimation du vent par exemple).
- L'agent *gcs* est l'interface graphique, il affiche les informations venant de *server* et renvoie les ordres de l'opérateur.

[4]Le projet publie les sources du code et du matériel sous license libre.

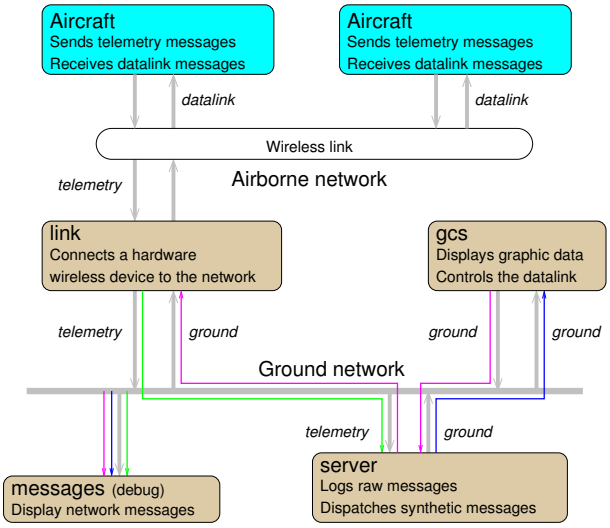

FIG. 7 – Communications entre les agents de la station sol Paparazzi.

Cette architecture permet d'ajouter des fonctionnalités au système simplement en connectant de nouveaux agents au bus *Ivy*.

Les agents de notre architecture réactive se greffent au système par l'intermédiaire de l'agent *environnement*. Ce dernier récupère sur le bus Ivy les données émises par l'agent *server* afin de maintenir l'état de la patrouille de drones et envoie à l'agent *link* les actions à effectuer.

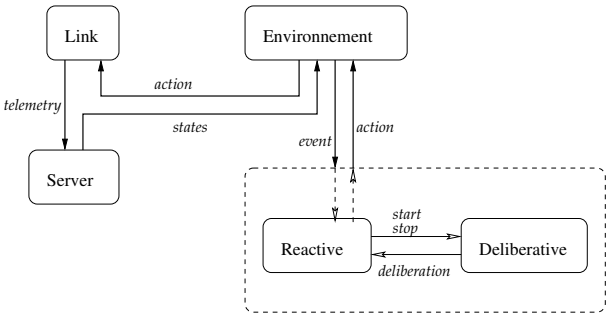

FIG. 8 – Intégration des agents de l'architecture réactive au système Paparazzi.

6 Résultats

6.1 Comparaison des deux méthodes de résolution

Afin de pouvoir tester les deux méthodes de résolutions proposées nous avons généré des instances de mission telles que celles des compétitions MAV, IMAV ou EMAV. Les positions des cibles sont générées aléatoirement. On note ces instances MAV-*X*-*Y* où X est le nombre de drones et Y le nombre de cibles.

Pour chaque instance nous avons fait tourner chacune de ces deux méthodes ainsi :

– la méthode de séparation évaluation jusqu'à prouver l'optimalité de la meilleure solution trouvée. Cela nous permet de connaitre la solution optimale.
– On arrête la méthode de recherche locale après 1 minute (temps minimum d'une action dans les instances testées).

Nous avons rapporté les résultats dans le tableau 1. La colonne OT représente le temps mis par la méthode de séparation évaluation pour trouver la solution optimale alors que ER est le temps pour prouver

l'optimalité (finir la recherche). La colonne LT est le temps mis par la méthode de recherche locale à grand voisinage pour atteindre la meilleure solution trouvée en moins d'une minute, et L/C le rapport entre le coût de la solution obtenu en 1 minute par la recherche locale et le coût de la solution optimale.

Instance	OT	ER	LT	L/C
MAV2-08	0.2s	0.2s	2.2s	100%
MAV3-08	0.1s	0.2s	0.7s	100%
MAV4-08	0.1s	0.3s	0.2	100%
MAV2-12	7s	8s	9s	104%
MAV3-12	0.7s	5.6s	3s	100%
MAV4-12	4.5s	25s	3.9s	100.5%
MAV2-14	110s	130s	56s	100%
MAV3-14	33s	145s	3.4s	102%
MAV4-14	126s	464s	6s	108%
MAV2-16	9min	10min	15s	101.6%
MAV3-16	55min	80min	5.6s	100.6%
MAV4-16	169min	220min	50s	101%

TAB. 1 – Résultats des méthodes de séparation et évaluation et recherche locale à grand voisinage

La figure 9 montre la différence des vitesses de convergence des 2 méthodes présentées. Pour les instances difficiles (avec plus de 14 cibles) la méthode locale à grand voisinage converge plus vite vers une bonne solution qui est souvent très proche de l'optimale comme on peut le voir sur le tableau 1. On remarque aussi que les deux méthodes présentées trouvent une première solution extrèmement vite (moins de 1 seconde). La qualité des solutions trouvées augmente rapidement au début de la recherche pour les deux méthodes, ce qui les rend très intéressantes à utiliser en tant qu'algorithme anytime.

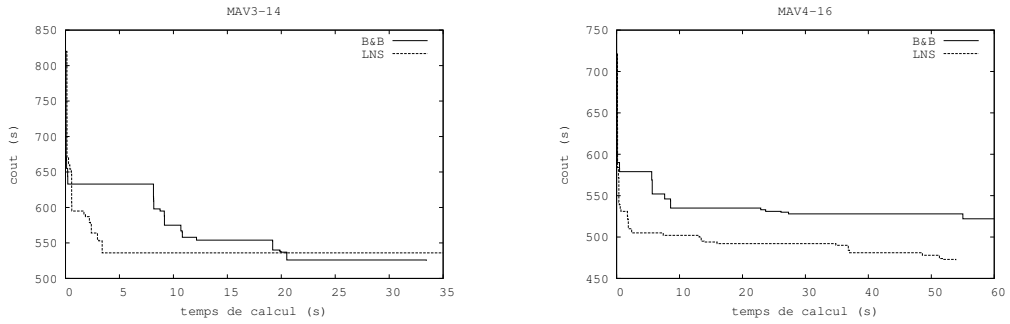

FIG. 9 – Comparaison entre méthode de séparation évaluation (B&B) et méthode de recherche locale à grand voisinage (LNS) sur les instances MAV3-14 et MAV4-16

6.2 Simulation en ligne

Pour tester notre système il faut créer des scénarios d'exécution définissant les événements qui vont se produire pendant la mission. Nous présentons ici les résultats obtenus à l'aide du simulateur du logiciel Paparazzi sur une mission avec une localisation, deux largages et sept identifications à réaliser ; les trois drones que nous avons simulés étaient des drones d'une envergure de 33cm pouvant chacun larguer une bille.

Nous avons comparé trois approches pour le système réactif-délibératif. La première, purement réactive, se contente d'utiliser les règles du système de décision réactif sans jamais planifier les actions à faire. La seconde utilise la planification hors ligne puis le système réactif. Enfin la dernière, décrite dans la section précédente, hybride planification en ligne et système réactif.

Voici les différents scénarios de tests :

- *Scénario 0* : l'exécution de la mission se déroule de façon nominale, il n'y a pas d'incertitude. Ce scenario décrit une exécution où tout se déroule comme prévu.
- *Scénario 1* : l'identification de deux des cibles échoue une fois.
- *Scénario 2* : l'identification de trois des cibles échoue une fois, l'identification d'une autre cible et la localisation échouent quatre fois chacune.
- *Scénario 3, 4 & 5* : identiques aux scenarios 0, 1 et 2 mais on a ajouté le crash du drone numéro 3 après 150s.

Nous avons rapporté les résultats des simulations dans le tableau 2. Pour chaque simulation nous avons noté les temps d'exécution de la mission de chacun des drones, et mis en gras la durée maximale (ou durée de la mission). Par exemple pour le *scénario 3* avec le système réactif le premier drone a mis 707s pour effectuer la mission, le deuxième drone a mis 568s et le troisième drone s'est crashé pendant la mission, ce que l'on note —.

L'approche hybride réactive délibérative que nous présentons a de meilleurs résultats que les deux approches réactives auxquelles nous la comparons sauf pour le *scénario 5*. En effet le *scénario 5* (plutôt catastrophique) montre les limites de la planification en ligne avec des hypothèses optimistes (sans prise en compte des incertitudes). Le faible écart que l'on trouve entre les trois méthodes sur le scénario 5 s'explique par le fait que les délibérations calculées par le planificateur doivent être remises en question très souvent, c'est la partie réactive qui prend alors le pas sur la partie délibérative.

Scenarios	Réactif	Hors Ligne + R	En ligne + R
Scenario 0	431 416 **747**	*461* ***487*** *487*	*461* ***487*** *487*
Scenario 1	452 **621** 589	377 554 **600**	***592*** *567 566*
Scenario 2	602 **1216** 772	**940** 796 722	***907*** *718 903*
Scenario 3	**707** 568 —	649 **690** —	*567* ***611*** —
Scenario 4	669 **773** —	673 **821** —	***629*** *592* —
Scenario 5	1174 **1229** —	*1171* ***1205*** —	1038 **1248** —

TAB. 2 – Résultat des simulations sur la mission de de test.

7 Conclusion

Dans cet article, nous avons présenté un système de planification de mission en ligne. Nous avons modélisé le problème sous forme de problème de satisfaction de contraintes et écrit un planificateur s'appuyant sur ce modèle grâce au solveur FaCiLe. Nous avons ensuite intégré ce planificateur dans le système Paparazzi en utilisant une approche hybride entre système réactif et délibératif. Les tests réalisés montrent clairement l'intérêt de cette approche hybride réactive et délibérative pour palier au mieux aux incertitudes lors de l'exécution de la mission.

Dans la suite de nos travaux nous envisageons d'aborder les problèmes dues à l'utilisation d'un planificateur optimiste afin de prendre en compte les échecs possibles dans la phase de planification.

Références

BAPTISTE P., PAPE C. L. & NUIJTEN W. (2001). *Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems*. Kluwer's International Series.

BARNIER N. (2002). *Application de la programmation par contraintes à des problèmes de gestion du trafic aérien*. PhD thesis, Institut National Polytechnique de Toulouse.

BECK J. C., PROSSER P. & SELENSKY E. (2003). Vehicle routing and job shop scheduling : What's the difference ? In *International Conference on Automated Planning and Scheduling*.

BRISSET P., DROUIN A., GORRAZ M., HUARD P.-S. & TYLER J. (2006). The Paparazzi solution. In *MAV2006*, Sandestin, Florida.

BUISSON M., BUSTICO A., CHATTY S., COLIN F.-R., JESTIN Y., MAURY S., MERTZ C. & TRUILLET P. (2002). Ivy : un bus logiciel au service du développement de prototypes de systèmes interactifs. In *IHM*

'02 : Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine), p. 223–226, New York, NY, USA : ACM.

CASEAU Y. & LABURTHE F. (1997). Solving small TSPs with constraints. In *ICLP*, p. 316–330.

DEAN T. & BOBBY M. (1988). An analysis of time-dependent planning. In *AAAI*, p. 49–54.

LEMAÎTRE M. & VERFAILLIE G. (2007). Interaction between reactive and deliberative tasks for on-line decision-making. In *ICAPS07 : International Conference on Automated Planning and Scheduling 2007*, Providence, Rhode Island, USA.

MUSCETTOLA N., MORRIS P. & TSAMARDINOS I. (1998). Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*, p. 444–452.

OSTERGAARD E., MATARIC M. & SUKHATME G. (2001). Distributed multi-robot task allocation for emergency handling. In *Intelligent Robots and Systems (IEE)*, Maui, HI, USA.

ROUSSEAU L.-M., GENDREAU M. & PESANT G. (2002). Solving small VRPTWs with constraint programming based column generation. In *Proceedings CPAIOR'02*.

SHAW P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, p. 417–431 : Springer-Verlag.

ZILBERSTEIN S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, **17**, 73–83.